



**TALLINN UNIVERSITY OF TECHNOLOGY**  
SCHOOL OF ENGINEERING  
Department of Electrical Power Engineering and Mechatronics

**DRONE MOVING TARGET LANDING SYSTEM AND  
ALGORITHM FOR SEMI- AND FULL AUTONOMOUS  
LANDING CONTROL**

**DROONI LIIKUVAL PLATVORMIL MAANDUMISE SÜSTEEM  
JA ALGORITM POOL- NING TÄISAUTONOOMSE  
MAANDUMISE JUHTIMISEKS**

MASTER THESIS

Student: ANASTASIA KRUTYAKOVA

Student Code: 196645MAHM

Supervisors: Mart Tamre  
Alexander Kapitonov

Tallinn 2020

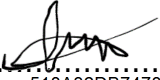
## AUTHOR'S DECLARATION

Hereby I declare, that I have written this thesis independently.

No academic degree has been applied for based on this material. All works, major viewpoints and data of the other authors used in this thesis have been referenced.

"....." ..... 2020 .

Author: Anastasia Krutyakova

DocuSigned by:  
  
.....  
518A32DB747849C...  
/signature /

Thesis is in accordance with terms and requirements "....." ..... 2020 .

Supervisors: Mart Tamre

.....  
/signature/

Alexander Kapitonov

DocuSigned by:  
  
.....  
5F60D98BA6124C0...  
/signature/

Accepted for defence

".....".....2020 .

Chairman of theses defence commission: .....

/name and signature/

# Non-exclusive Licence for Publication and Reproduction of Graduation Thesis<sup>1</sup>

I, Anastasia Krutyakova, (date of birth: 12.11.1996) hereby

1. grant Tallinn University of Technology (TalTech) a non-exclusive license for my thesis Drone moving target landing system and algorithm for semi- and full autonomous landing control,

*(title of the graduation thesis)*

supervised by Professor Mart Tamre, Associate Professor Alexander Kapitonov,

1.1 reproduced for the purposes of preservation and electronic publication, incl. to be entered in the digital collection of TalTech library until expiry of the term of copyright;

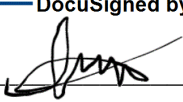
1.2 published via the web of TalTech, incl. to be entered in the digital collection of TalTech library until expiry of the term of copyright.

1.3 I am aware that the author also retains the rights specified in clause 1 of this license.

2. I confirm that granting the non-exclusive license does not infringe third persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

---

<sup>1</sup> *Non-exclusive Licence for Publication and Reproduction of Graduation Thesis is not valid during the validity period of restriction on access, except the university's right to reproduce the thesis only for preservation purposes.*

DocuSigned by:  
 (signature)  
518A32DB747849C...

5/25/2020 (date)



# TABLE OF CONTENTS

AUTHOR'S DECLARATION.....	2
Non-exclusive Licence for Publication and Reproduction of GraduationTthesis <sup>1</sup> .....	3
THESIS TASK.....	4
TABLE OF CONTENTS.....	5
LIST OF FIGURES .....	7
LIST OF TABLES .....	8
LIST OF ABBREVIATIONS .....	9
INTRODUCTION.....	10
1. LITERATURE OVERVIEW .....	12
1.1. Drone aircraft lands on roof of car travelling at 75 km/h .....	12
1.2. Quadrotor landing on moving platform .....	13
1.3. Drone delivery systems .....	15
1.3.1. Amazon Prime Air .....	15
1.3.2. Uber Eats Drone .....	16
1.3.3. Wing .....	17
1.4. Conclusion .....	17
2. ALGORITHM OVERVIEW.....	19
2.1. Description of the algorithm performance.....	19
2.2. Adverse conditions .....	20
2.3. Performance during adverse conditions.....	20
2.4. Conclusion .....	21
3. DEVELOPMENT OF THE ALGORITHM .....	23
3.1. Software installation.....	23
3.1.1. Description of rospackages in the workspace .....	24
3.2. Graphical interface.....	26
3.2.1. Start menu.....	26
3.2.2. Flight management menu .....	27
3.2.3. Manual control menu .....	28
3.2.4. Landing menu.....	29

3.2.5. Emergency menu .....	29
3.3. Landing module .....	30
3.3.1. Tracking-Landing Algorithm.....	31
3.3.2. Detection and Localization of the Mobile Platform .....	32
3.3.3. Tracking the Mobile Platform .....	33
3.4. Manual control module .....	34
3.5. Emergency module .....	37
3.6. Conclusion .....	38
4. SIMULATION .....	39
4.1. Design of the Testing Environment.....	39
4.2. Testing the system.....	40
4.2.1. Performing the landing on the moving platform.....	42
4.2.2. Testing the Emergency Mode.....	45
4.2.3. Testing the Manual Control Mode .....	46
4.3. Conclusion .....	47
5. CONCLUSIONS AND FUTURE RECOMMENDATIONS.....	48
5.1. Conclusion .....	48
5.2. Future recommendations .....	48
REFERENCES .....	50
APPENDICES .....	53
Appendix 1 The start menu .....	53
Appendix 2 The start launch file .....	54
Appendix 3 The flight management menu .....	56
Appendix 4 The landing mode file .....	61
Appendix 5 The landing mode launch file .....	62
Appendix 6 The manual mode file .....	63
Appendix 7 The manual mode launch file.....	64

## LIST OF FIGURES

Figure 1.1 Penguin BE landed on roof of car travelling at 75kmph [4] .....	12
Figure 1.2 Amazon's delivery drone [3] .....	15
Figure 1.3 Uber Eats Drone [2] .....	16
Figure 1.4 Wing's delivery drone [17] .....	17
Figure 2.1 The flowchart of the developed algorithm .....	19
Figure 3.1 The Start Menu .....	26
Figure 3.2 The Flight Management Menu .....	28
Figure 3.3 The Manual Control Menu .....	29
Figure 3.4 The Landing Menu .....	29
Figure 3.5 The Emergency Menu.....	30
Figure 3.7 Pinhole camera geometry [27] .....	33
Figure 3.8 The PlayStation 4 Joystick [29] .....	34
Figure 3.8 "jstest-gtk" .....	37
Figure 4.1 The designed simulated environment.....	39
Figure 4.2 The Gazebo simulation and "Flight Management" menu both opened by "start.py" .....	40
Figure 4.3 ROS nodes and ROS topics created.....	40
Figure 4.4 The Gazebo simulation and "Landing" menu after performed landing .....	42
Figure 4.5 ROS nodes and ROS topics created.....	43
Figure 4.6 The Gazebo simulation and "Landing" menu after performed landing .....	45
Figure 4.7 The Gazebo simulation and "Manual Control" menu .....	46

## **LIST OF TABLES**

Table 3.1 PS4 Controller Map. Buttons .....	35
Table 3.2 PS4 Controller Map. Axes. ....	36



## **LIST OF ABBREVIATIONS**

UAV	Unmanned Aerial Vehicle.
ROS	Robot Operating System
EKF	Extended Kalman Filter
UGV	Unmanned Ground Vehicle
PID	Proportional-Integral-Derivative
3D	Three-Dimensional
IBVS	Image-Based Visual Servo
MPC	Model Predictive Control
VTOL	Vertical Take-Off and Landing
HSV	Hue, Saturation, Value
IMU	Inertial Measurement Unit
MAE	Mean Absolute Error
CSV	Comma-Separated Values
GUI	Graphical User Interface
URDF	Unified Robot Description Format

## INTRODUCTION

Over the past few decades, robotics has become one of the most promising spheres. Automatization of manufacturing processes makes it possible to perform production faster and cheaper, mobile robots have turned up to be an enormous help in rescuing and military missions and companion robots are to become extremely useful assistants. Unmanned aerial vehicles (UAV) or drones have numerous civilian, commercial, military, and aerospace applications. The drones have been operated in the delivery business for a long time now but there are still many issues unsolved.

First of all, drone cannot operate in high winds or rain. Another challenge is manoeuvring: the vast majority of deliveries would need to be in an urban environment where potential airborne threats are considerably higher than rural areas. The most significant challenge is that the use of airborne delivery systems requires a significant amount of approval. Not only from boards and engineers, but from aviation authorities across the world. Any aircraft in the world requires flight plans and safety checks before they are allowed to leave the runway. However using drones in delivering is quite promising thing as they can be implemented for food delivery along with medicine, mails and etc. To make that possible not only mentioned problems must be solved but also there should be no laws broken and UAVs should run as much autonomously as possible.

One of the best solutions of drone-based delivery in the cities is making it possible to UAVs to perform a landing on roofs of moving cars. It'll make a landing process safer, prolong the distance and the speed that drone can reach, help avoid non-flying zones and etc. There also may be some places where it's really difficult to perform a drone landing and it'll provide an ability for people to catch a drone with a delivered package. [1] There are some companies that are really interested in performing a drone landing on a moving target such as Uber [2] or Amazon [3]. Their researches will be overviewed in Chapter 1.

This work is dedicated to solving the problem of detecting the opportunity for a drone to land on a target placed on a roof of a moving platform and performing the autonomous landing if it is possible. The work will include both theoretical and practical part. Theoretical part will include overview of existing solutions for an autonomous drone landing systems, autonomous drone landing systems on moving targets and of companies that already use drone delivery systems or about to implement one. Practical part will include designing a simple algorithm for automatic landing, implementation of the algorithm in C++ and Python, designing a graphical user interface (GUI) to control the system and testing the algorithm under "Gazebo" simulator. ROS will be used as

the communication framework between the whole set of components in the architecture. The operative system utilized for running the different processes is Ubuntu 16.04.

In this work will be presented the algorithm that will allow drone to make the landing decision. If the landing couldn't be performed drone needs to understand the reason why it can't be performed (weather conditions, high speed of the landing platform, losing the target). Other important thing is that drone has to understand the remaining charge of the it's battery. If there are no landing platforms and drone is not able to fly any more it should perform emergency landing on a safe ground and send its own coordinated so the operator could understand that there is a problem and rescue the drone.

## 1. LITERATURE OVERVIEW

This chapter describes existing methods and solutions for drone landing: (1) drone aircraft landing on top of a car travelling at 75 km/h (Section 1.1); (2) brief description of different researches on quadrotor landing on a moving platform (Section 1.2); (3) brief description of drone delivery services (Section 1.3) and (4) concluding part (Section 1.4).

### 1.1. Drone aircraft lands on roof of car travelling at 75 km/h

In 2016 team of researcher from German Aerospace Center landed [4] an unmanned, electric-powered, autonomous aircraft on top of a car travelling at 75 km/h. Test flight involved a three-meter, 20 kg fixed-wing UAV Penguin BE. The landing was achieved using optical targets on the roof of the car, which was equipped with a flat net for the UAV to rest on. As the aircraft approached the car, it used the targets to line itself up and gauge distance within 50 cm, bringing itself in for approach and landing under computer control. This landing technique can be useful when applied to solar-powered drones operating at altitudes of over 20 km for weeks at a time, by simplifying landings in adverse weather conditions.



Figure 1.1 Penguin BE landed on roof of car travelling at 75 km/h [4]

## 1.2. Quadrotor landing on moving platform

There are quite a few papers dedicated to an autonomous quadrotor landing. They all use different control techniques but all more or less solve the problem of drone landing on a moving target. Almost all works use Kalman filter.

The algorithm [5] presents the design of a landing platform and its relative pose estimation. Two filters were designed to conduct the fusion: an EKF and an Extended  $H_{\infty}$  ( $EH_{\infty}$ ). Simulation was done in Gazebo. The experimental tests were conducted on the AR.Drone 2.0 quadrotor.

The system [6] was experimentally validated by successfully landing in multiple trials a commercial quadcopter on the roof of a car moving at speeds of up to 50 km/h. System architecture was presented, including the structure of the Kalman filter for the estimation of the relative position and velocity between the quadcopter and the landing pad, as well as the controller design for the full rendezvous and landing manoeuvres. The algorithms were implemented in C++ using ROS. Also, several real-flight experiments were done outdoors. As it was said before, UAV performed landing satisfyingly on the platform with the speed up to 50 km/h.

Paper [7] also solves the problem of autonomous drone landing on a moving target. Usually, algorithms are based on tracking the desired target, but in this paper, approach is based on other three stages: estimation, prediction and fast landing. Simulation was done in Gazebo. All the software components were developed using C++ and ROS as the communication framework between the whole set of components in the architecture.

Paper [8] proposes a vision-based neural network controller for the autonomous landing of a quadrotor on moving targets. The landing is performed with an error of less than 37 cm from the centre of a mobile platform traveling at a speed of up to 12 m/s under the condition of noisy measurements and wind disturbances. To simulate the developed controller the ROS framework was used along with Gazebo. The controller was implemented on the AR.Drone 2.0 quadrotor. No real-life experiments were made.

Landing system [9] is using model predictive control (MPC). To provide the position, velocity and acceleration of the UAV the EKF was used. The approach consists of vision-based target following, optimal target localization, and model predictive control, for optimal guidance of the UAV. To validate the control method for autonomous landing on a moving platform, the DJI hardware-in-the-loop (HITL) simulation environment was used. To connect all the components of the system ROS framework was used. Since the

UAV, that was used in the work, can fly at a speed of up to 18 m/s, the maximum target speed set in the simulations was 12 m/s for a headwind speed of 5 m/s.

System [10] uses a composite landmark for landing on a moving platform. The landing system contains four layers: sensor layer, data fusion layer, decision layer and control layer. The sensor layer consists of a variety of heterogeneous sensors, including the R2D landmark, encoder, inertial measurement unit (IMU), and GPS. The sensor layer provides the measurements of the moving platform and drone. The fusion layer uses the measurement data from the sensor layer to obtain the real-time pose estimation of the UAV and the moving platform. The decision layer generates the desired waypoint and trajectory online based on the real-time pose estimation from the data fusion layer. The control layer realizes the visual tracking of the moving platform and the pose control of the UAV. EKF is used to deal with temporarily missing visual information, the unknown measurement bias of encoders caused by wheel-slip and imprecise calibration is taken into consideration in the target's dynamical model, and the state of the moving platform. The Gazebo and ROS simulation software were used to build the experimental environment. The real-flight experiments were successfully done outdoors.

System [11] is an autonomous landing system that uses only onboard sensing and computing. Communication between modules happens through ROS. To deal with missing visual detections, as well as to estimate the full state of the platform (namely the position, velocity and orientation), EKF was used. Gazebo and RotorS were used to validate the framework in simulation. A Clearpath Husky UGV simulated model as ground vehicle, on top of which the detected tag was mounted. In the real-world experiments a Clearpath Jackal was used as ground vehicle carrying the landing platform and was controlled manually. In nominal conditions the platform can reach a maximum speed of 2 m/s. A 150x150 cm wooden landing pad equipped with the tag was installed on the top of the vehicle, reducing its maximum speed to approximately 1,5 m/s due to the additional weight. Experiment was performed successfully indoors.

Paper [12] suggests to eliminate the need for a landing gear by landing on a mobile ground vehicle. This would not only increase the payload capacity, but also simplify landings in crosswind conditions and thus increase the operational availability. A system with a small UAV and a car-mounted landing platform was prepared as a technology demonstrator. Different aspects of the landing problem were studied in simulations and real experiments and algorithms for the cooperative control of both vehicles were proposed.

Paper [13] does not present any landing system but it describes a robust localization system. The localization algorithm is done resorting to odometry and global vision data fusion, applying an EKF.

There are also a couple of patent dedicated to solving the drone landing problem such as [14], [15] and [16]. It is another proof of importance of such systems.

### **1.3. Drone delivery systems**

As mentioned before, delivery systems are most likely to be interested in suggested solution. There are some companies that provide delivery serviced, that have already started to implement drones into their delivery processes.

#### **1.3.1. Amazon Prime Air**

Amazon Prime Air is a project made by Amazon [3]. It is a drone-based delivery system. Amazon has presented different designs of drones but working principal is similar: customer choses a product, Amazon operator put the product in the box and attaches this box to a drone. Drone has simple grabbers, which will not let the box to fall off during the flight. Landing point should be marked by specific QR-marker, so the camera under the drone will be able to detect it.



Figure 1.2 Amazon's delivery drone [3]

Still the drone is able to carry only light and compact goods (maximum 2,5 kg), but the delivery process would be fully autonomous and would not take much time. The delivery should be performed in no longer than 30 minutes. For example, the first test delivery was performed in 13 minutes. This is why the company might be interested in using cars, because in real life 30-minute delivery is really fast. Moreover, it will help to increase the weight of box that drone is carrying.

### **1.3.2. Uber Eats Drone**

Uber Eats is an exciting delivery project that is done by Uber. In 2019 during the Forbes 30 under 30 Summit the drone's design was unveiled [2].



Figure 1.3 Uber Eats Drone [2]

This drone can carry dinner for up two people and, featuring rotating wings with six rotors, the vehicle can vertically take-off and land, and travel a maximum of eight minutes, including loading and unloading. The total flight range is 18 miles (28,9 km), with a round-trip delivery range of 12 miles (19,3 km). This mean that the drone can do up to three six-mile legs: up to 6 miles to the restaurant, up to 6 miles from the restaurant to the customer and up to 6 miles back to its launch area.

Uber's plan is to fly meals from restaurants to a staging location where an Uber driver would then travel the last mile for the hand-off to the customer. It has also considered landing drones on the roofs of delivery cars, the company said in June when it unveiled its drone testing plans. At the time, Uber Eats had made a few test deliveries from a McDonald's near San Diego State University.



### 1.3.3. Wing

Wing, an Alphabet company, has built a small, lightweight aircraft and navigation system that can deliver small packages – including food, medicine and household items – directly to homes in minutes. Created in 2012, Wing has conducted more than 80,000+ flights across three continents [17]. With an expanded Air Carrier Certificate from the Federal Aviation Administration (FAA), Wing today became the first company to operate a commercial air delivery service via drone directly to homes in the United States.



Figure 1.4 Wing's delivery drone [17]

Wing's drone hovers at about 24 toes and lowers a package deal to the bottom with a tether. The drone seems extra like a small aircraft. Its two wings, extending greater than three toes, every characteristic a propeller and make allowance the drone to fly additional whilst holding power, the corporate says. Altogether, the drone has 14 propellers designed to cut back noise.

## 1.4. Conclusion

As it is seen from Section 1.3, delivery companies start to think how to implement drones into their delivery processes. UAV will not only be able to perform the whole delivery process, but they can be useful to minimize the average delivery time.

There are several things that people expect from delivery companies [18]: speed of delivery, size and weight limitations, proof of delivery, customer service, cost to value and courier insurance. Using drones in delivery will not only increase the speed of delivery, it can also reduce the cost of delivery and minimize the human factor.

There are lots of papers that solve the problem of the autonomous drone landing on moving platforms, but there are some issues that don't allow to implement those algorithms in real life:

- 1) Most of the real-life experiments were done indoors and with low-speed landing pads.
- 2) Experiments, that were done outdoors, were performed during the day and with sunny conditions.

Designing a system, that will be able not only to perform the autonomous drone landing on a moving platform but also have an ability to detect whether the landing can be performed or not, will help to make the landing algorithms actual implementations dealing with the scenario in which the landing target is non-stationary.

## 2. ALGORITHM OVERVIEW

This chapter describes the proposed algorithm: (1) general description of the algorithm (Section 2.1); (2) naming the adverse conditions that can affect the landing process (Section 2.2); (3) general description of the algorithm performance affected by adverse conditions (Section 2.3) and (4) concluding part (Section 2.4).

### 2.1. Description of the algorithm performance

The UAV gets a command to perform the landing on the moving target. After receiving a command, the drone should check if the landing can be performed. Adverse conditions are described in Section 2.2. If there are no adverse conditions, the drone will perform the landing and send the message to the operator. The landing process is described in Section 3.3. If any of the adverse conditions is present, the drone sends a warning message to the operator. After getting the warning message operator can take control over the drone and make a decision of what to do. If operator doesn't take the control, the drone will perform the "emergency landing" on the ground and send the operator another message with its coordinates. A flowchart, describing the algorithm, is presented on Fig. 2.1.

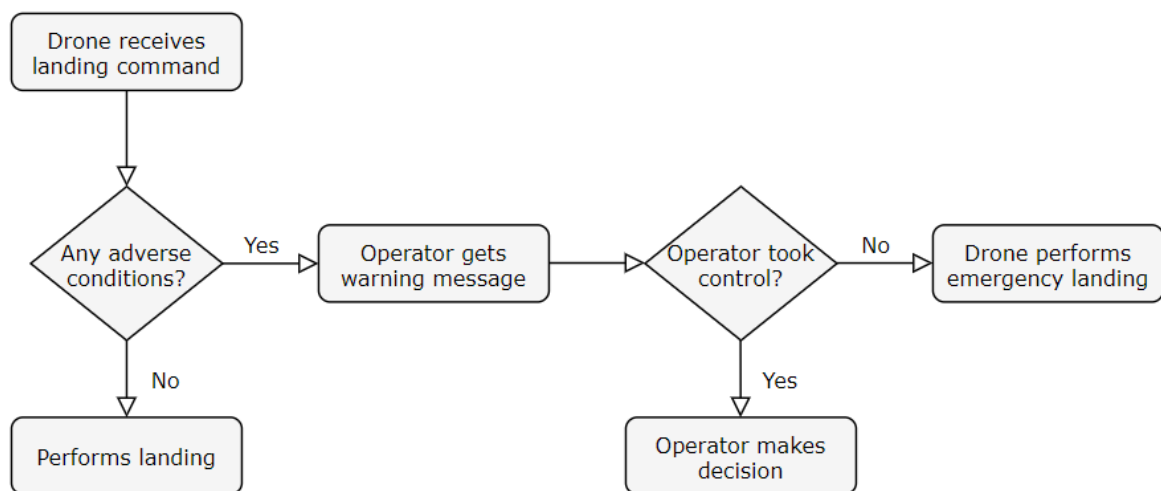


Figure 2.1 The flowchart of the developed algorithm

## **2.2. Adverse conditions**

There are different conditions that can affect the precision of the landing process or make it completely impossible. In general, those conditions can be divided into two groups: weather conditions and technical conditions.

Weather conditions include the speed of the wind, the temperature, the level of humidity. The exact numbers are model dependent and have to fulfil the recommendations in the drone's datasheet. The high speed of the wind can not only make the landing process impossible, but it can also make the flight process very dangerous (on average, absolute speed of 12 m/s is considered to be the critical wind speed). The low temperature cause the rapid discharge of the UAV battery. Moreover, some elements of the plastic structure become brittle and also change their volume and dimensions due to imminent cooling, which in theory can lead to mechanical malfunctions. High humidity, as well as snow or heavy rain, can damage the drone (water drops can penetrate inside the UAV, for example, into the motors) [19].

Technical conditions include the remaining charge of the drone's battery, the availability of required onboard sensors. At the beginning of the landing, it is suggested to have the remaining charge of the drone's battery no lower than 40%. The landing on the moving pad takes more time than landing on the static one. The low level of the battery charge can not only force the shutdown of the drone, but it also can be responsible for the inaccuracy in the sensor's work. Therewith, before performing the landing the UAV has to make sure that all its sensors are working properly and providing the drone with all the required information.

## **2.3. Performance during adverse conditions**

If any of the adverse conditions reveal itself during the flight, the option of the landing on the moving platform becomes unavailable and the option of emergency landing becomes available.

Operator gets to choose from a couple of options: switch to the manual control, make the drone perform the emergency landing or make the drone continue the flight. However some of the adverse conditions cause the emergency landing even if the operator doesn't choose this option (for example, low level of battery charge).

Manual mode allows the operator to control the UAV actions and movements via controller or keyboard. This mode allows the operator to perform the landing manually.

Emergency mode lands the quadrotor on the ground and sends the drone coordinates to the operator. It is possible to switch the emergency mode to the manual mode (but it is not recommended) or to continue the flight in autonomous mode.

However, the operator can miss the warning message sent by the drone, so to minimize the human factor it was decided to add the autonomous decision-making algorithm. Each adverse condition gets a flag, that identifies the level of danger that the condition can cause, and these flags will set the behaviour of the drone. For example, all the conditions were divided into three groups: first ones get "Flag #1", the second ones get "Flag #2" and the last ones get "Flag #3".

If the condition has occurred and it has "Flag #1", the drone will perform the emergency landing and keep sending the error message with the drone's coordinates to the operator. If the condition has "Flag #2", the drone will continue the flight and keep sending the message to the operator that the condition has been occurred, but the drone has continued its route. If the condition has "Flag #3", the drone will continue the flight. But it also will keep checking if the condition is still present, and after 3 tries the drone will make the final decision. If the condition is still present, then the drone will make the emergency landing, otherwise the drone will continue the route.

## **2.4. Conclusion**

The general idea of the algorithm is to make the operating process as autonomous as possible. This chapter only describes the algorithm, the programming process is described in Chapter 3. The adverse conditions described in Section 2.2 can be changed if needed. The described algorithm can be implemented for any drone model. The described algorithm should be able to do the following:

1. To perform the landing on the moving platform when it is required.
2. To monitor the flight of the drone.
3. To check if the landing can be performed safely.
4. To make the control process easier for operators.
5. To add a decision-making part.

The algorithm is designed to be easy-to-implement. Many of the algorithms, described in Chapter 1, can be implemented with only specific drone models. However, this algorithm should be developed to be able to work on any drone model. But the UAV should fit in some requirements:

1. The drone should have a bottom camera: no high-resolution is needed.
2. The drone should have IMU sensor.
3. The drone should have an onboard GPS module.

If any of the mentioned requirements is not met, then the algorithm will not work properly.

## 3. DEVELOPMENT OF THE ALGORITHM

This chapter describes the process of the development of the algorithm: (1) description of software installation process (Section 3.1); (2) design of the graphical interface (Section 3.2); (3) description of the landing module (Section 3.3), (4) description of the manual control module (Section 3.4), (5) description of the emergency module (Section 3.5) and (6) concluding part (Section 3.6).

### 3.1. Software installation

ROS is a robotic programming framework that provides functionality for distributed work. ROS provides standard operating system services, such as: hardware abstraction, low-level device control, the implementation of frequently used functions, message passing between processes and packet management. ROS is based on graph architecture, where data processing occurs at nodes that can receive and transmit messages between themselves. The library is focused on Unix-like systems [20].

There are two ROS versions that are still supported at the beginning of year 2020: ROS Kinetic and ROS Melodic. It is also possible to install earlier versions, but it is suggested to use the supported one. It was decided to use ROS under Ubuntu. Ubuntu is an operating system based on Debian GNU / Linux. Ubuntu also has versions. But certain ROS versions can be easily installed only under certain Ubuntu version. That is why it was decided to choose ROS version at first, and after that choose the Ubuntu version.

ROS Kinetic is the tenth ROS distribution release. It was released May 23rd, 2016 and is still supported. The release date is the main reason, why it was chosen to use this version: ROS has a huge amount of different packages, each package solves the problem either partially or completely and packages are usually available for certain versions. ROS Kinetic is primarily targeted at the Ubuntu 16.04 (Xenial) release. ROS Kinetic also uses Python 2. The desktop image allows to try Ubuntu without changing the computer at all and to install it permanently later. The image can be downloaded from the official website [21] and installed using instructions given in [22]. After successful installation of Ubuntu it is time to install ROS. The instructions are given in [23]. It is suggested to install the full version as it will install Gazebo and other useful packages as well.

After installing ROS it is required to create a workspace using command "catkin\_make" (catkin was installed with ROS Kinetic). There is an instruction given in [24]. All the

rospackages installed from GitHub and packages created during this work are put in created workspace. Also it is needed to install some python modules, C++ libraries and ROS packages. The Python modules that are needed: Tkinter, Roslaunch, Rospypy.

ROS packages that are used in this work are described in Appendix A. Also two ROS projects were installed from GitHub [25] and [26] and were used in this work. [25] is a project that allows to control a drone using ROS. It describes messages, actions and other thing that will be very useful. [26] is a project that performs the autonomous landing on the moving platform and it will be used as a base for performing the landing.

### 3.1.1. Description of rospackages in the workspace

**ardrone\_moves package.** This package allows to control the drone moves. The package contains "ardrone\_moves.cpp" that describes the movements of the drone. The described function listens to the rostopic "keyboard\_sub" and publishes data to rostopics "pub\_takeoff", "pub\_land", "pub\_toggle\_state" and "pub\_vel". This package also has a launch file "ardrone\_moves.launch" that creates two nodes: "teleop" using "teleop\_twist\_keyboard" .

**ped\_traj\_pred package.** This package describes the controllers that are used in the landing part. This package also describes how the landing module tracks the landing platform.

**summit\_moves package.** This package allows to control the Summit XL robot, that holds the landing platform. The package contains "summit\_moves.cpp" that describes the movements of the robot. This package also has a launch file "summit\_moves.launch" that creates the node "summit\_moves" and includes "ardrone\_teleop.launch" from the "takeoff" package.

**takeoff package.** This package describes the "Take Off" state of the system. It also contains the launch file for the world with both the UGV and the UAV.

**uav\_vision package.** This package describes the landing platform detecting algorithm.

**hector\_quadrotor\_actions package.** This package allows a drone to perform some basic actions. The package has three C++ files: "takeoff\_action.cpp", "pose\_action.cpp" and "landing\_action.cpp". "Landing\_action.cpp" – describes just simple landing on the ground. Each of these C++ files calls a header file "base\_action.h" that is also kept in



this package. This package also has a launch file "actions.launch" that creates three nodes: "pose\_action", "landing\_action" and "takeoff\_action".

**hector\_quadrotor\_controller\_gazebo package.** This package describes the quadrotor and its sensors in Gazebo. The package contains C++ file "quadrotor\_hardware\_gazebo.cpp". It calls a header file "quadrotor\_hardware\_gazebo.h" that is also kept in this package.

**hector\_quadrotor\_controllers package.** This package describes attitude, position and velocity controllers. The package contains three C++ files: "attitude\_controller.cpp", "position\_controller.cpp" and "velocity\_controller.cpp". This package also has a launch file "controller.launch" that declares arg "controllers", two rosparams (using files "controller.yaml" and "params.yaml" that are kept in this package) and creates two nodes: "controller\_spawner" and "estop\_relay".

**hector\_quadrotor\_description package.** This package provides a URDF model of a generic quadrotor UAV. The visual geometry is provided as a COLLADA model and the collision geometry is provided as a STL mesh.

**hector\_quadrotor\_gazebo package.** This package provides a quadrotor model based on hector\_quadrotor\_urdf that is usable in gazebo.

**hector\_quadrotor\_model package.** This package provides libraries that model several aspects of quadrotor dynamics.

**hector\_quadrotor\_pose\_estimation package.** This package estimates the 6DOF of a robot based on the EKF of various sensor sources.

**hector\_quadrotor\_teleop package.** This package provides a convenient gamepad-based control option for quadrotor UAVs and similar vehicles. The stick setup in the launch files is similar to the "Mode 2" setup commonly used on RC helicopters as described here. The node publishes geometry\_msgs/Twist messages on the /cmd\_vel topic, so the stick input corresponds to desired linear and angular velocities.

**hector\_uav\_msgs package.** This package is a message package that contains messages for UAV controller inputs and outputs and some sensor readings not covered by sensor\_msgs.

**interface package.** This package has GUI menus and launch files for the developed algorithm.

## 3.2. Graphical interface

It was decided to design a GUI to make the controlling process easier to operators and minimize the number of possible mistakes (like launching the wrong file). There are many different GUI-programming toolkits for Python, however, it was decided to use Tkinter module for Python 2.7.

This section describes the developed menu windows: (1) the "Start" menu (Item 3.2.1), (2) the "Flight Management" menu (Item 3.2.2), (3) the "Manual Control" menu (Item 3.2.3), (4) the "Landing Mode" menu (Item 3.2.4) and (4) the "Emergency Mode" menu.

### 3.2.1. Start menu

The "Start" menu is the first window to be opened. The menu can be opened by running the "start.py" file, that is kept in the "interface" package. This window gives the user two options: "Quit" and "Start". Clicking the "Quit" button will close the menu and kill all the process, that were started by the algorithm, by calling the created function "evCancel". Clicking the "Start" button will start the algorithm and open the next menu. The created function "mCreateManagement" will be called. This function launches the "start.launch" file and runs the "management.py" file. The files, that are launched using the "start.launch" file, will be described in Section 4.1 of Chapter 4. The GUI of this menu is presented on Fig. 3.1.

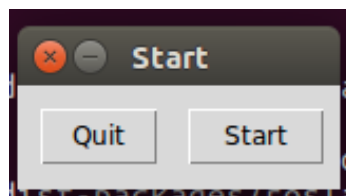


Figure 3.1 The Start Menu

### 3.2.2. Flight management menu

The "Flight Management" menu is opened when the user clicks the "Start" button in the "Start" menu. The "Flight Management" menu is used to control the flight and monitor different parameter, that can affect the landing as well as the flight itself. The parameters can be easily changed in the "management.py" file. Every parameter gets the data using messages, that created ROS nodes are publishing into certain ROS topics. The required data is gotten by subscribing to the required ROS topics. For every parameter the range of values is set. The values that are not fitting in this range are considered as triggers to start the emergency landing. To change or add the parameter it is needed to change/add the text label block and to change/add the message block in the "management.py" file. The text label block has the name of the parameter and the message block contains the information about the message that has the required data and the ROS topic, where the message has been published.

Also, the "Flight Management" menu gives the user four options: "Quit", "Manual Mode", "Landing Mode" and "Emergency Mode". Clicking the "Quit" button will close the menu and kill all the process, that were started by the algorithm, by calling the created function "evCancel". Clicking the "Manual Mode" button will allow to control the drone manually using a controller or keyboard. The "Manual Control" menu will be opened. The created function "mCreateManual" will be called. This function launches the "manual.launch" file and runs the "manual.py". The files, that are launched using the "manual.launch" file, will be described in Section 3.4. Clicking the "Landing Mode" button will trigger the landing on the moving platform. The "Landing Mode" menu will be opened. The created function "mCreateLanding" will be called. This function launches the "landing.launch" file and runs the "landing.py". The files, that are launched using the "landing.launch" file, will be described in Section 3.3. Clicking the "Emergency Mode" button will trigger the landing on the moving platform. The "Emergency Mode" menu will be opened. The created function "mCreateEmergency" will be called. This function launches the "emergency.launch" file and runs the "emergency.py". The files, that are launched using the "emergency.launch" file, will be described in Section 3.5. The function "mCreateEmergency" is also called when the emergency landing is triggered when any of the parameter's value is not fitting in the set range. The GUI of the "Flight Management" menu is presented in Fig. 3.2.

The parameters can be set in the beginning of the "management.py" using function "rospy.set\_param('[name]', value)" or by getting the existing one by using function "rospy.get\_param('[name]')". In a case, when the parameter can't be gotten from the simulation environment (like wind speed in Gazebo 7), then it can be set manually. The

parameter, that was set manually, can still be changed by running "roscparam set [name] new\_value".

Also the function "refreshlabel" was created to refresh parameters values. It gets the new value of the parameter every second. The function is called in the description of the label.

The "management.py" file is presented in Appendix 3.

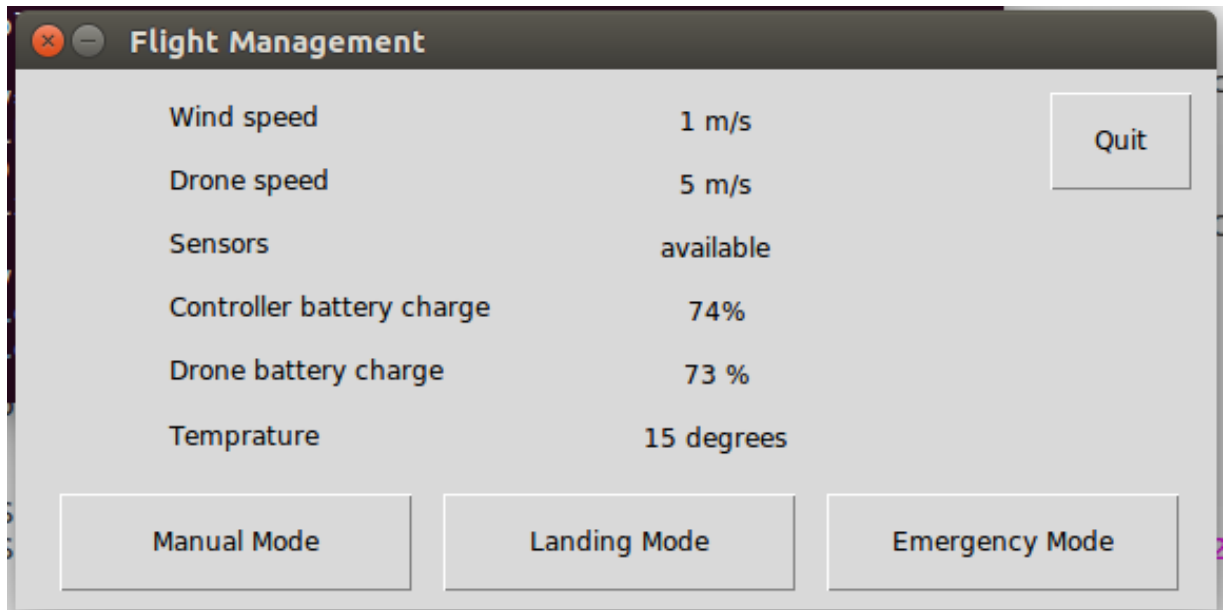


Figure 3.2 The Flight Management Menu

### 3.2.3. Manual control menu

The "Manual Control" menu is opened when the user clicks the "Manual Mode" button in either the "Flight Management" menu, the "Landing Mode" menu or the "Emergency Mode" menu. The "Manual Control" menu has the only option - "Quit". Clicking the "Quit" button will close the menu and kill all the process, that were started by the algorithm, by calling the created function "evCancel". The GUI of the menu is presented in Fig. 3.3.

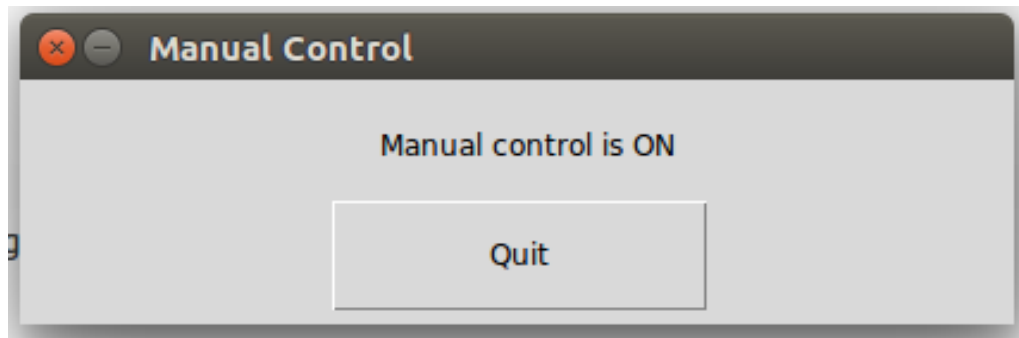


Figure 3.3 The Manual Control Menu

### 3.2.4. Landing menu

The "Landing Mode" menu is opened when the user clicks the "Landing Mode" button in the "Flight Management" menu. The "Landing Menu" gives the user two options: "Quit" and "Manual Mode". Clicking the "Quit" button will close the menu and kill all the process, that were started by the algorithm, by calling the created function "evCancel". Clicking the "Manual Mode" button will allow to control the drone manually using a controller or keyboard. The "Manual Control" menu will be opened. The created function "mCreateManual" will be called. This function launches the "manual.launch" file and runs the "manual.py". The GUI of the menu is presented in Fig. 3.4.

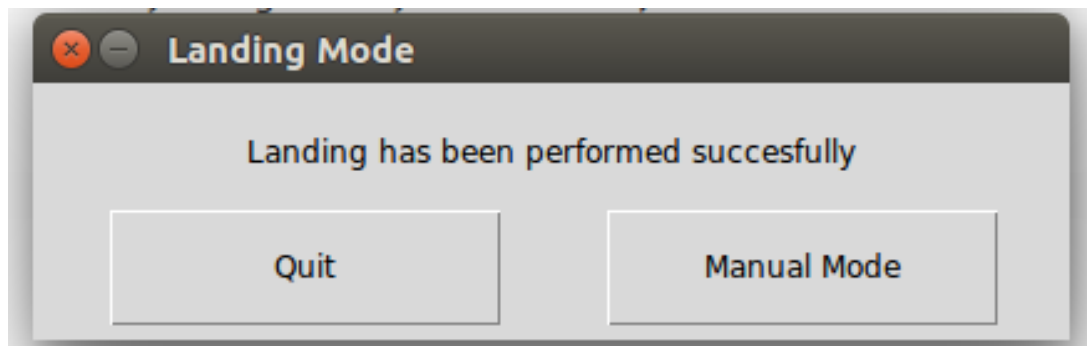


Figure 3.4 The Landing Menu

### 3.2.5. Emergency menu

The "Emergency Mode" menu is opened if the user clicks the "Emergency Mode" button or if the emergency landing has been triggered when any of the parameter's value is not fitting in the set range. The "Emergency Mode" menu has the information about the drone's coordinates. The coordinates are published in the topic

"/ardrone/ground\_truth/state". The message type is Odometry from "nav\_msgs". Also the "Emergency Mode" menu gives the user two options: "Quit" and "Manual Mode". Clicking the "Quit" button will close the menu and kill all the process, that were started by the algorithm, by calling the created function "evCancel". Clicking the "Manual Mode" button will allow to control the drone manually using a controller or keyboard. The "Manual Control" menu will be opened. The created function "mCreateManual" will be called. This function launches the "manual.launch" file and runs the "manual.py". The GUI of the menu is presented in Fig. 3.5.

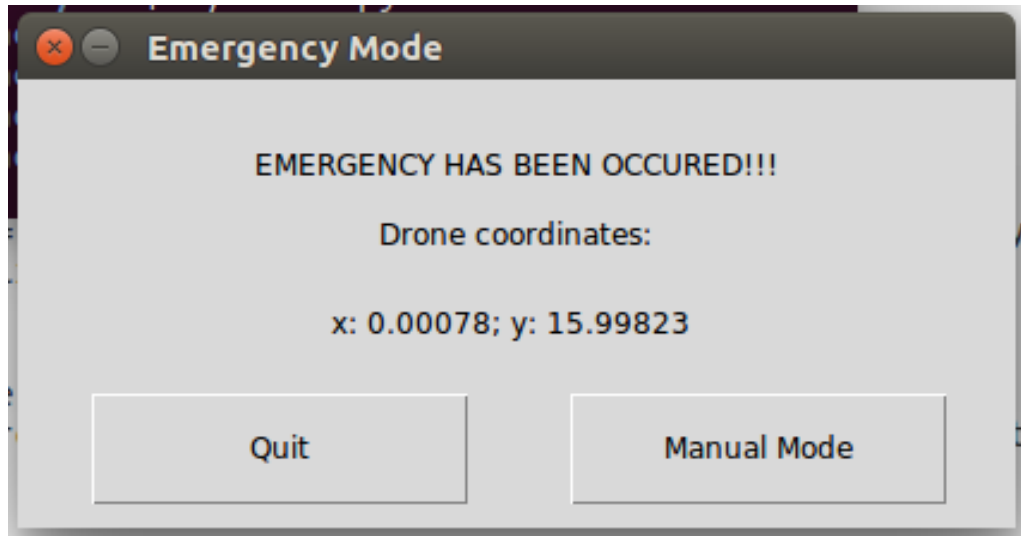


Figure 3.5 The Emergency Menu

### 3.3. Landing module

The landing module is executed by the "landing.launch" file. This file launches the "platform\_detection.launch" file, the "platform\_tracking.launch" file and the "kalman\_pred.launch" file. The "platform\_detection.launch" file starts the node called "platform\_detection" using the "platform\_detection.cpp" file. The "platform\_detection.cpp" file will be described in Item 3.3.2. The "platform\_tracking.launch" file starts the node called "platform\_tracking" using the "platform\_tracking.cpp" file. The "platform\_tracking.cpp" file will be described in Item 3.3.3. The "kalman\_pred.launch" file starts the node called "prediction\_kalman\_node" using the file "ped\_traj\_pred.cpp". The "ped\_traj\_pred.cpp" will be described in Item 3.3.3. The landing platform that is being detected is presented in Fig. 3.6.

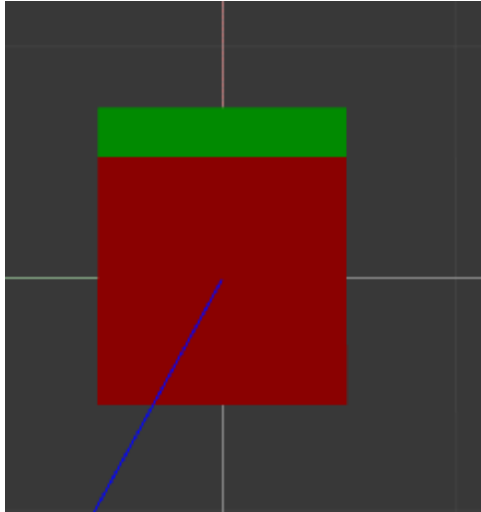


Figure 3.6 The landing platform

The landing module is based on [26]. This section described the proposed algorithm in details: (1) the tracking-landing algorithm based on a finite state machine (Item 3.3.1), (2) the algorithm of detecting and localizing the moving landing platform (Item 3.3.2) and (3) the tracking algorithm (Item 3.3.3).

### 3.3.1. Tracking-Landing Algorithm

The tracking-landing algorithm proposed in [26] is a finite state machine, that has 5 states: "Taking Off", "Tracking", "Landing", "Relocating" and "Landed". "Landed" is the default state in [27], but for proposed algorithm it is not always true, as the landing module is just a part of the system. When the system is launched by clicking the "Start" button in the "Start" menu, the drone is landed on the landing platform and has a state "Landed". The state "Landed" will change to "Taking Off" when the "Landing Mode" is chosen. If the user doesn't choose any of the options in the "Flight Management" menu for 30 seconds, then the drone will take off automatically. If this happens, the state will be changed to "Taking Off". The reason why the state is "Taking Off" and not "Tracking" is that before performing the landing part drone has to reach the nominal height (that is set to 4 meters in the experiment).

So, after clicking the "Landing Mode" in the "Flight Management" menu, the drone starts to gain/lower altitude with a constant speed of 1 m/s along its z axis. Despite the fact, that the state is "Taking Off", the detection-localization algorithm and the tracking algorithm are launched at this point. It makes it possible for a drone to start detecting

and following the platform while reaching the nominal height. The detection-localization algorithm is described in Section 3.3.2 and the tracking algorithm is described in Section 3.3.3.

As the nominal height has been reached, the state changes to "Tracking" and the drone stops changing the height. To make the drone follow the landing platform keeping the required altitude PID controller is used. After 30 seconds of tracking the state will shift to "Landing" and the landing manoeuvre will start. The drone will start to reduce the height with a constant speed of 0.3 m/s along its z axis. Another PID controls the speed of the drone along the x and y axis.

### **3.3.2. Detection and Localization of the Mobile Platform**

In order to detect and localize the landing platform OpenCV was used. There are lots of different options to detect a landing pad, but here was decided to use simple colour- and shape-detection, even though the most of the researches described in Chapter 1 used AprilTags and other markers to detect.

The algorithm for detecting and localizing the moving landing platform is presented in the "platform\_detection.cpp" file. The algorithm is working the following way:

1. The image is gotten from the video frame using the ROS topics.
2. The input frame is converted into the HSV colour model using "cvtColor" function.
3. All the pixels that correspond to the red colour are kept by applying a colour mask using "inRange" function.
4. The Canny algorithm is used to detect the edges using "findContours" function.
5. The Hough Line transform algorithm is used to detect straight lines using "line" function.
6. "approxPolyDP" function is used to check the lines to find a polygone.
7. The centroid's coordinates are calculated using functions "Moments" and "Point2f".

The centroid's coordinates are used to compute the 3D coordinates of this centroid. To do so the pinhole inverse transformation (pinhole camera model) is used. This function is described in [27]. The pinhole camera model is illustrated in Fig. 3.7.



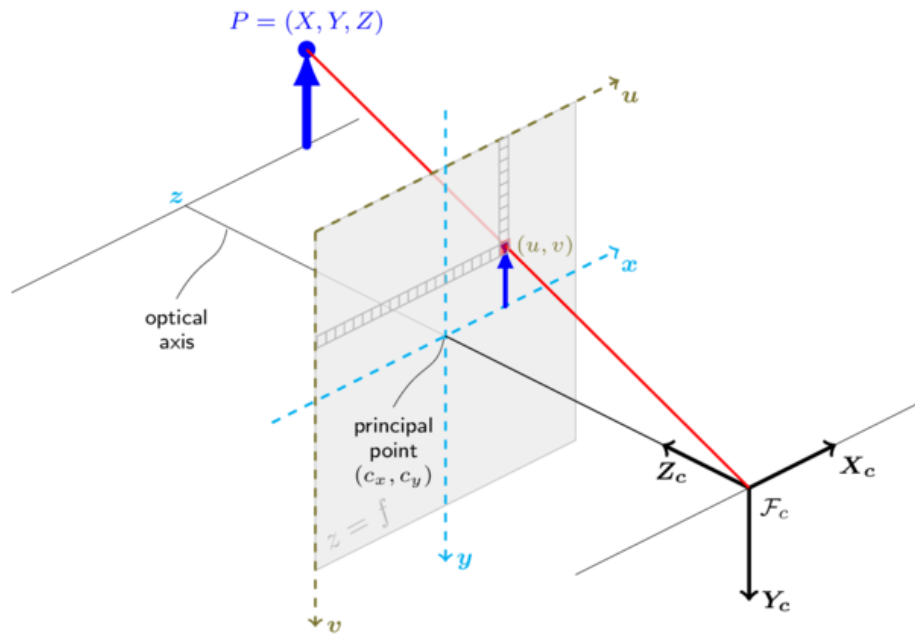


Figure 3.7 Pinhole camera geometry [27]

### 3.3.3. Tracking the Mobile Platform

To track the moving landing platform the pedestrian trajectory prediction algorithm [28] was used. This algorithm is presented in the "ped\_traj\_pred.cpp" file. The position that was calculated in Item 3.3.2 is sent to this prediction algorithm. This algorithm returns a vector of future positions of the centroid of the landing platform relative to the world's frame. The first element in this vector corresponds to the current position of the moving platform. The second element in this vector corresponds to the next position of the moving platform (after a 0,1 seconds). The third element in this vector corresponds to the position of the moving platform after 0,2 seconds (twice the defined time step).

The predicted future position of the landing platform is transformed from the world's frame into the UAV's body frame. Finally, the  $x$  and  $y$  coordinates of UAV's body frame are used to calculate the controller's error, i.e., the distance in the  $xy$ -plane between the UAV and the predicted position of the landing platform. Using this error, the speed commands in  $x$  and  $y$  can be calculated. This will make the controller's error converge to zero. This method is described in [26] in Section 3.3.2.

### 3.4. Manual control module

To perform the manual control PlayStation4 Joystick is used. It is much easier than controlling the drone using the keyboard.



Figure 3.8 The PlayStation 4 Joystick [29]

To connect the joystick to the laptop it is required to install ds4drv from GitHub. There are two ways to connect the joystick to the laptop: via Bluetooth or via USB. Buttons on the joystick are coded and each button has a certain code. The button codes that correspond to joystick buttons are presented in Table 3.1. Table 3.2 describes axes that can be controlled by the joystick and corresponded buttons and sticks on the controller.

Table 3.1 PS4 Controller Map. Buttons

<b>Nr</b>	<b>Button code</b>	<b>Joystick button</b>
1	joystick button 0	Square
2	joystick button 1	X
3	joystick button 2	Circle
4	joystick button 3	Triangle
5	joystick button 4	L1
6	joystick button 5	R1
7	joystick button 6	L2
8	joystick button 7	R2
9	joystick button 8	Share
10	joystick button 9	Options
11	joystick button 10	L3
12	joystick button 11	R3
13	joystick button 12	PS4 On Button
14	joystick button 13	Touch Pad Press

Table 3.2 PS4 Controller Map. Axes.

<b>Nr</b>	<b>Axis</b>	<b>Joystick button/stick</b>
1	X-Axis	Left Stick X
2	Y-Axis	Left Stick Y
3	Z-Axis	Right Stick X
4	4 <sup>th</sup> Axis	Right Stick Y
5	5 <sup>th</sup> Axis	L2
6	6 <sup>th</sup> Axis	R2
7	7 <sup>th</sup> Axis	Touch Pad X
8	8 <sup>th</sup> Axis	Touch Pad Y

The joystick can be tested via "jstest-gtk" command. "Jstest-gtk" can be installed with "apt-get" command. It is recommended to test the controller before running the algorithm. If everything is okay, the results should be close to the ones, that are presented on Fig. 3.8. Notice, that the port the controller has been assigned to have to match the one that is defined in "ardrone\_teleop.launch".

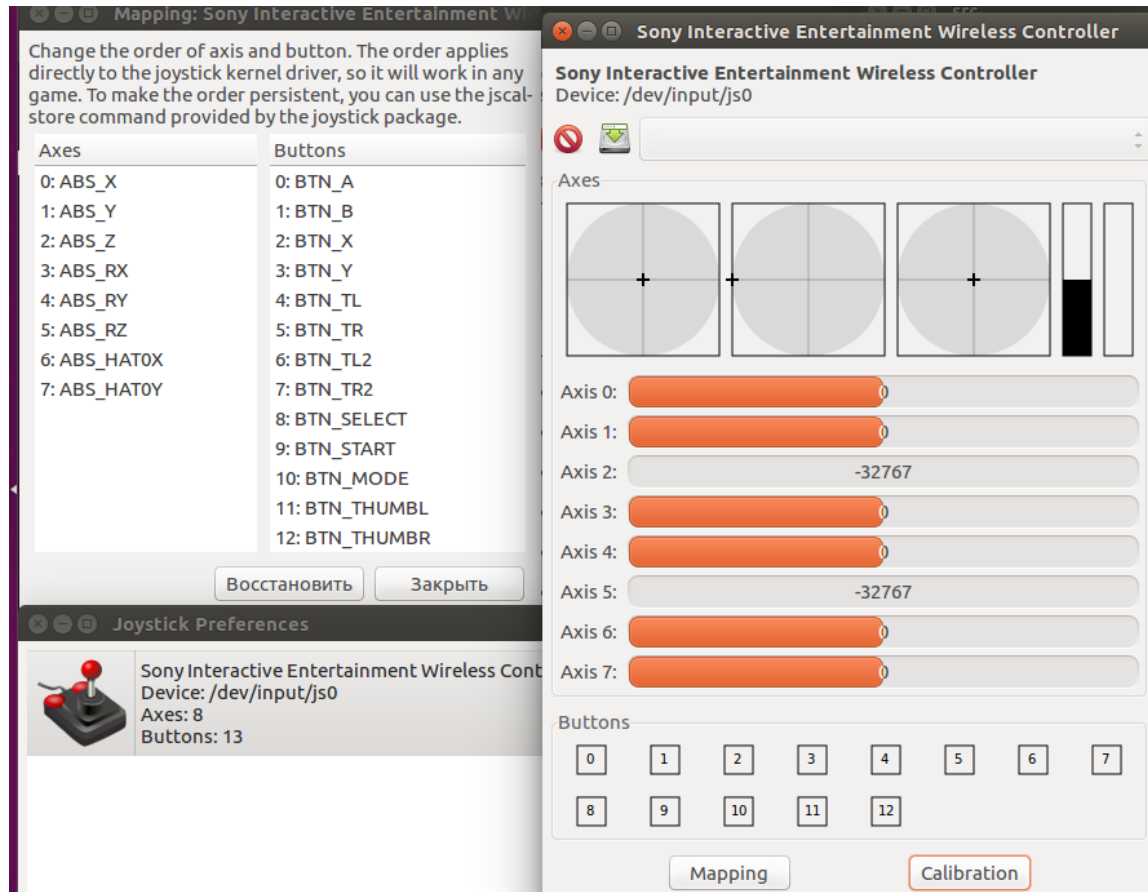


Figure 3.8 "jstest-gtk"

The created package "interface" has a launch file called "manual.launch" that enables the manual control by launching the "ardrone\_teleop.launch". The controller has to be connected before running the whole algorithm, otherwise the error will be occurred and the manual control won't be working. The function mCreateManual was created to begin the work of the manual control module. It launches "manual.launch" file and opens the "Manual Control" menu by running "manual.py" file.

### 3.5. Emergency module

Emergency module is actually a landing performing, except for sending the drones coordinates. The package "interface" has a launch file called "emergency.launch" that gives a command to the drone to perform the landing on the ground and forces the execution of the coordinates of the drone. The function mCreateEmergency was created to begin the work of the emergency module. It launches "emergency.launch" file and opens "Emergency Mode" menu by running "emergency.py" file.

There are two ways of beginning the "emergency" scenario. The first one is by pressing the "Emergency mode" button in the "Flight Management" menu. The second one is by provoking one of the triggers (adverse condition occurring). The triggers are set in "management.py" file, that sets the GUI of the "Flight Management" menu. Every measured parameter set in the file gets a minimum and maximum values. If the parameter changes itself to a value, that is not fitting those numbers anymore, then the function mCreateEmergency is called. The landing process is a part of basic actions and is described in "basic\_actions.h".

The drone is using ground truth information from Gazebo for localization that simulates the work of GPS module. The coordinates are published via PositionXYCommand. This message type is described in "PositionXYCommand.msg" file and has two data values of float32 type: "x" and "y".

### **3.6. Conclusion**

The algorithm was developed and the graphical interface was designed. One of the main ideas of the algorithm is to make the working process easier to the operator and minimize the human factor.

The designed GUI allows to the operator to operate with a drone only using pre-programmed buttons. And it also allows operator to see all the necessary information on their screen in a human-understanding way.

## 4. SIMULATION

This chapter describes the results in the simulated environment: (1) design of the testing environment (Section 4.1); (2) results of the tests (Section 4.2) and (3) concluding part (Section 4.3).

### 4.1. Design of the Testing Environment

The simulated environment was designed in Gazebo. A new Gazebo world was designed for this work. The "ground\_plane" model is placed with the coordinates (0,00; 0,00; 0,00). The "summit\_xl" model is placed on the "ground-plane" with the coordinates (0,00; 0,00; 0,00). The "summit\_xl" model has a landing platform attached. The "ardrone" model is placed on the "summit\_xl" with the coordinates (-0,01; 0,00; 0,53). The designed environment is presented in Fig. 4.1.

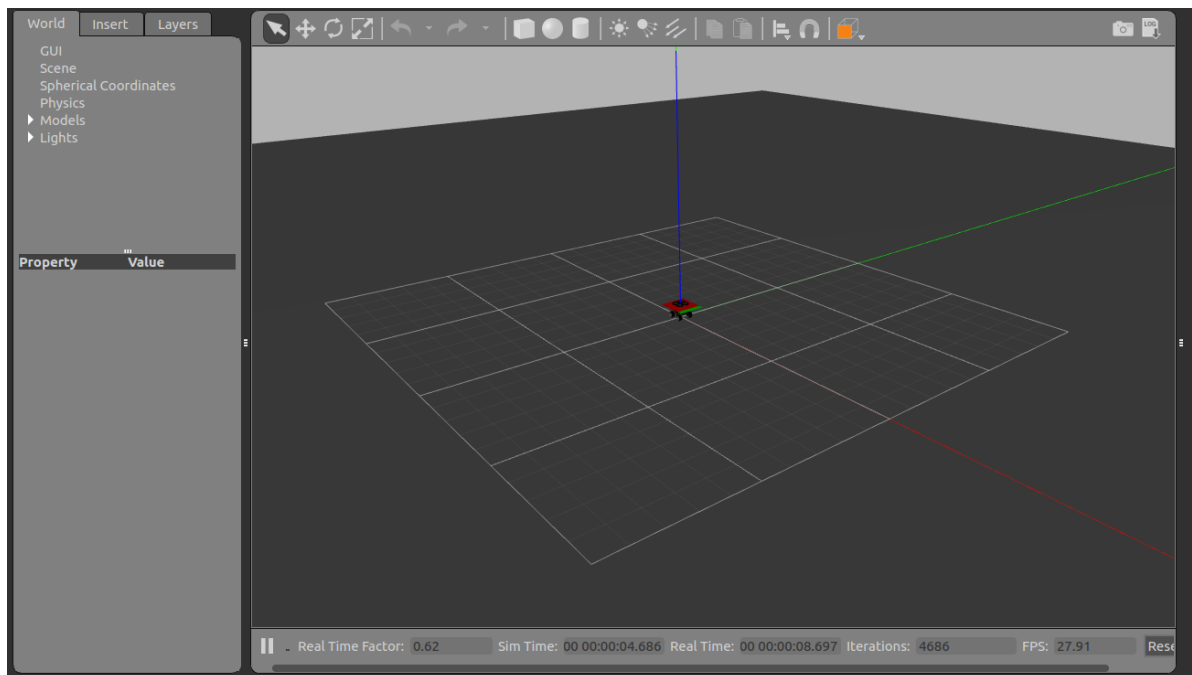


Figure 4.1 The designed simulated environment

## 4.2. Testing the system

Before starting the simulation it is required to source the "setup.bash" file in the "devel" folder in the root of the workspace. To begin the simulation the file "start.py" from the package "interface" should be launched. After pressing the start button the Gazebo simulation will be launched and the "Flight management" menu will be opened.

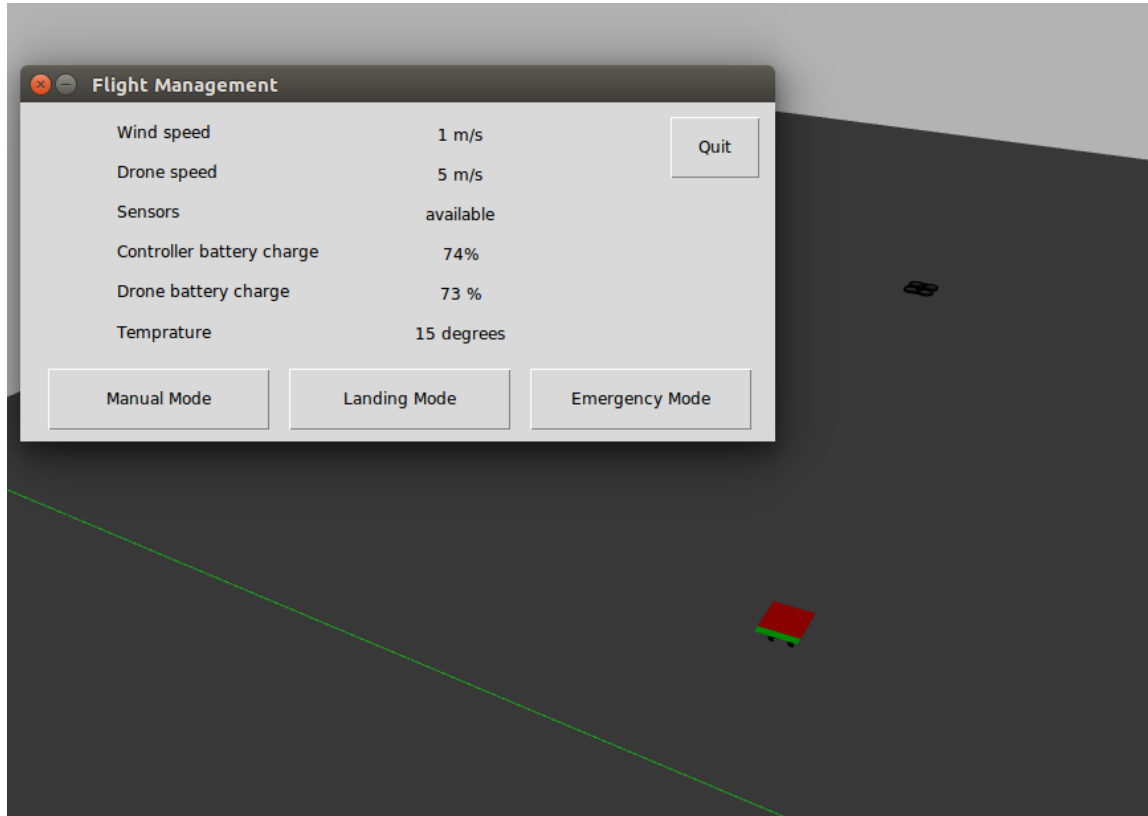


Figure 4.2 The Gazebo simulation and "Flight Management" menu both opened by "start.py"

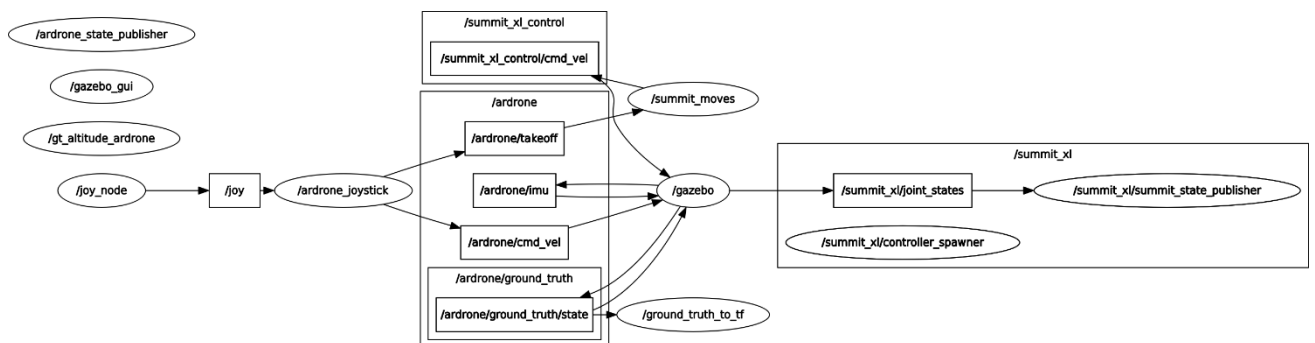


Figure 4.3 ROS nodes and ROS topics created



As it is seen from Fig. 4.3, the following nodes were created: "/ardrone\_state\_publisher", "/gazebo\_gui", "/gt\_atitude\_ardrone", "/joy\_node", "/ardrone\_joystick", "/summit\_moves", "/gazebo", "/ground\_truth\_to\_tf", "/summit\_xl/controller\_spawner" and "/summit\_xl/joint\_state\_publisher".

The following topics were created: "/joy", "/summit\_xl-control/cmd\_vel", "/ardrone/takeoff", "ardrone/imu", "ardrone/cmd\_vel", "ardrone/ground\_truth/state" and "summit\_xl/joint\_states".

The node "/joy\_node" publishes the data in gets from the joystick into the topic "/joy". The node "/ardrone\_joystick" gets this data from the topic "/joy" and publishes this data in two topics: "ardrone/cmd\_vel" and "/ardrone/takeoff". The node "/summit\_moves" gets the data from the "/ardrone/takeoff" topic and publishes it into the "/summit\_xl-control/cmd\_vel" topic. The "summit\_moves" is response for the Summit XL with the landing platform movements. The "/gazebo" node gets the information about the robot's movements from the "/summit\_xl-control/cmd\_vel" topic. The "/gazebo" node gets the information about the drone's movements from the "ardrone/cmd\_vel" topic, that also gets the information from the "/ardrone\_joystick" node. Moreover, the "/gazebo" node changes the information with "ardrone/imu" and "ardrone/ground\_truth/state" topics. The "ardrone/imu" topic contains the information about the drone's state. The "ardrone/ground\_truth/state" topic also provides information to the "/ground\_truth\_to\_tf" node, that is needed to coordinate the world's frame with the drone's and the robot's ones. The "gazebo" node also publishes messages into the "summit\_xl/joint\_states" topic. And the node "/summit\_xl/joint\_state\_publisher" is subscribed to this topic. The "summit\_xl/joint\_states" topic contains the information about Summit's joints.

### 4.2.1. Performing the landing on the moving platform

In the "Flight Management" menu the button "Landing Mode" is pressed. The drone reached the landing platform and performs the landing.

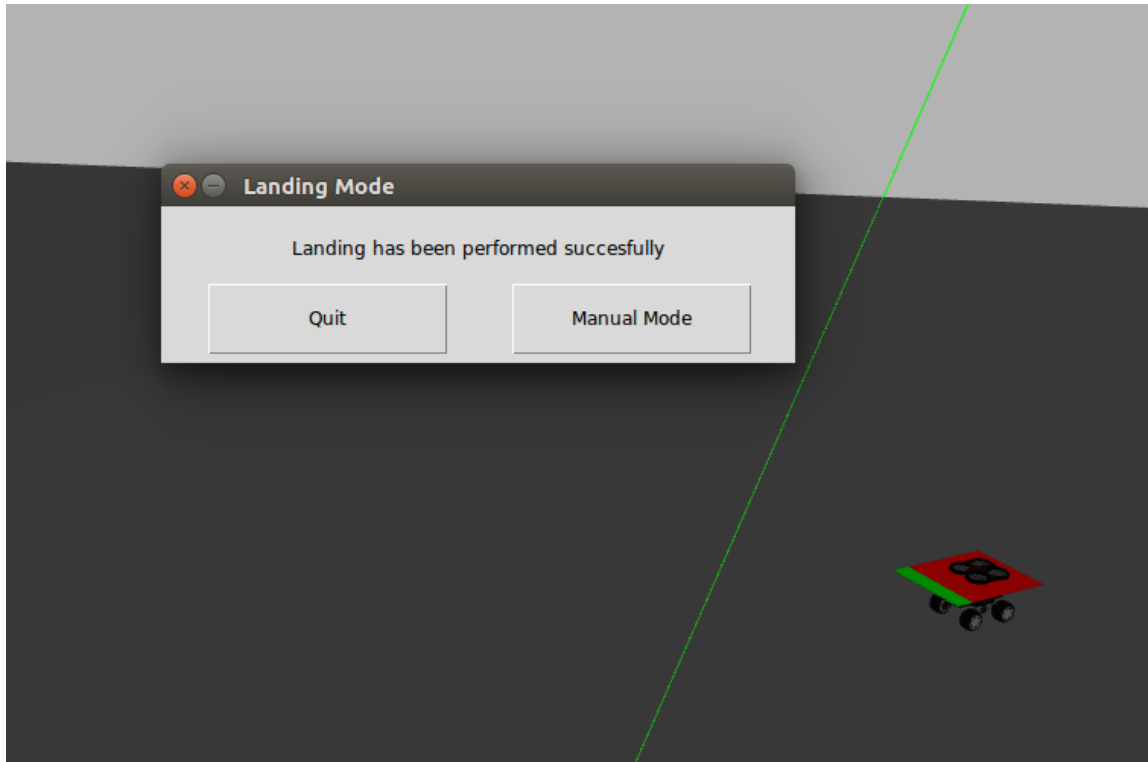


Figure 4.4 The Gazebo simulation and "Landing" menu after performed landing



in four topics: `"/ardrone/force_land"`, `"/ardrone/land"`, `"/ardrone/cmd_vel"` and `"/ardrone/takeoff"`. The node `"/summit_moves"` gets the data from the `"/ardrone/takeoff"` topic and publishes it into the `"/summit_xl-control/cmd_vel"` topic. The `"/summit_moves"` is response for the Summit XL with the landing platform movements. The `"/gazebo"` node gets the information about the robot's movements from the `"/summit_xl-control/cmd_vel"` topic. The node `"/platform_tracking"` also get the information from the `"/ardrone/takeoff"` topic, as well as from the `"/ardrone/force_land"` and `"/ardrone/land"` topics. The node `"/platform_tracking"` publishes information into the `"/ardrone/cmd_vel"`. The `"/gazebo"` node gets the information from `"/ardrone/ground_truth_state"` (and publishes information there at the same time), `"/ardrone/cmd_vel"`, `"/ardrone/imu"` (and publishes information there at the same time) and `"/ardrone/navdata"` topics. Alongside with the previously mentioned topics, the `"/gazebo"` node also publishes messages into the following topics: `"/summit_xl/joint_states"`, `"/ardrone/altimeter"`, `"/ardrone/sonar_height"`, `"/ardrone/bottom/image_raw"` and `"/ardrone/bottom/camera_info"`.

The `"/platform_detection"` node, which is created with the `"/platform_detection.cpp"` file, subscribes to the following topics: `"/ardrone/imu"`, `"/ardrone/bottom/image_raw"`, `"/ardrone/bottom/camera_info"`, `"/ardrone/altimeter"`, `"/ardrone/sonar_height"` and `"/ardrone/groundtruth_altitude"`. The results of the detection-localization algorithm (the landing platform's centroid coordinates) are published in the `"/platform/current_platform_position_in_world"` topic.

The `"/platform/current_platform_position_in_world"` topic shares this information with the `"/prediction_kalman_node"` node which runs the `"/ped_traj_pred.cpp"` file and published the vector of future positions of the landing platform into the `"/kalman_prediction_path"` topic. The `"/platform_tracking"` node is subscribed to the `"/kalman_prediction_path"` topic and uses this information to track the platform.

The `"/platform_tracking"` node is subscribed to the following topics: `"/ardrone/takeoff"`, `"/ardrone/imu"`, `"/ardrone/force_land"`, `"/ardrone/land"`, `"/ardrone/altimeter"`, `"/ardrone/sonar_height"`, `"/ardrone/groundtruth_altitude"`, `"/groundtruth/ardrone"`, `"/groundtruth/summit"` and `"/kalman_prediction_path"`. The `"/platform_tracking"` node not only tracks the landing platform, but also performs the landing.

### 4.2.2. Testing the Emergency Mode

In the "Flight Management" menu the button "Emergency Mode" is pressed. The drone starts to land on the ground. The Emergency menu is opened, that also shows the coordinates of the drone. The drone lands on the ground and it kept there until the "Manual Mode" is chosen. The results of this test are shown in Fig. 4.6. It was decided not to include the list of nodes and topics, as there are no special ones (emergency landing is being performed by calling the simple landing command, that is described in the ardrone controller description).

The "Emergency Mode" was also tested by turning the parameter "Drone Battery Charge" to the value of 10%. The emergency landing started automatically right after the parameter was changed, without clicking the "Emergency Mode" in the "Flight Management" menu.

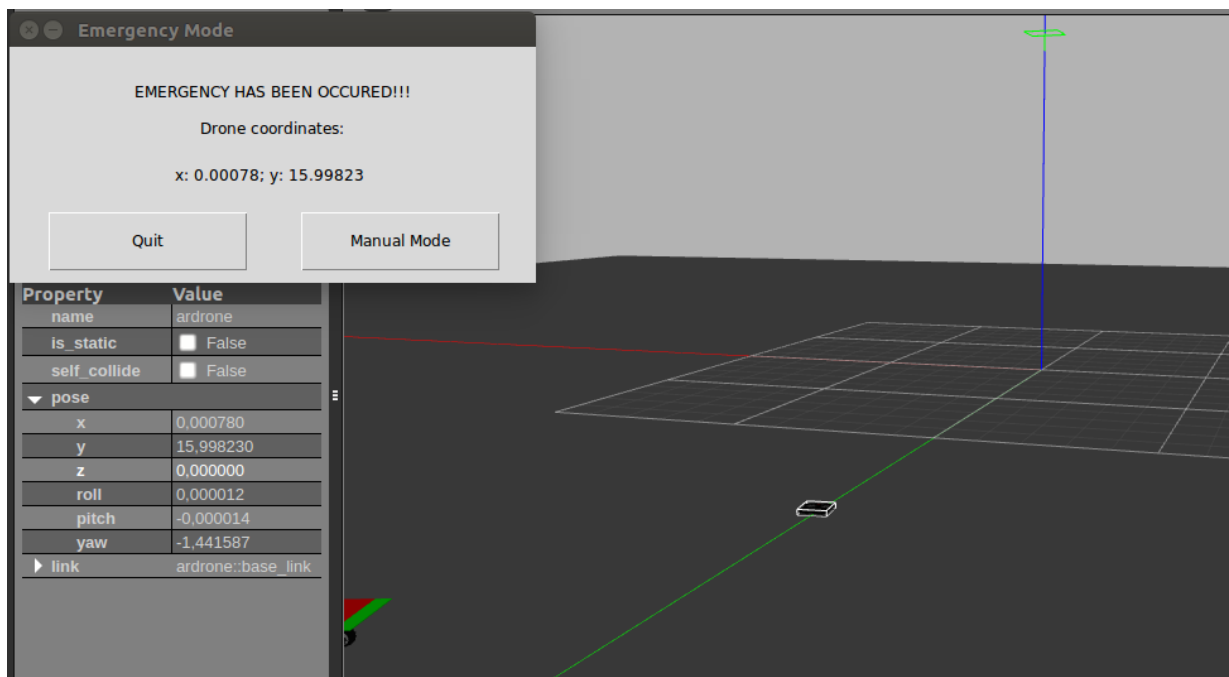


Figure 4.6 The Gazebo simulation and "Landing" menu after performed landing

### 4.2.3. Testing the Manual Control Mode

In the "Flight Management" menu the button "Manual Control" is pressed. The drone follows the command that are given through the PlayStation4 joystick. The robot with the landing platform continues to move, but the drone is fully controlled by the operator. It was decided not to include the list of nodes and topics, as there are no special ones (the node /ardrone\_joystick was created as the simulation started to force the Summit XL movements).

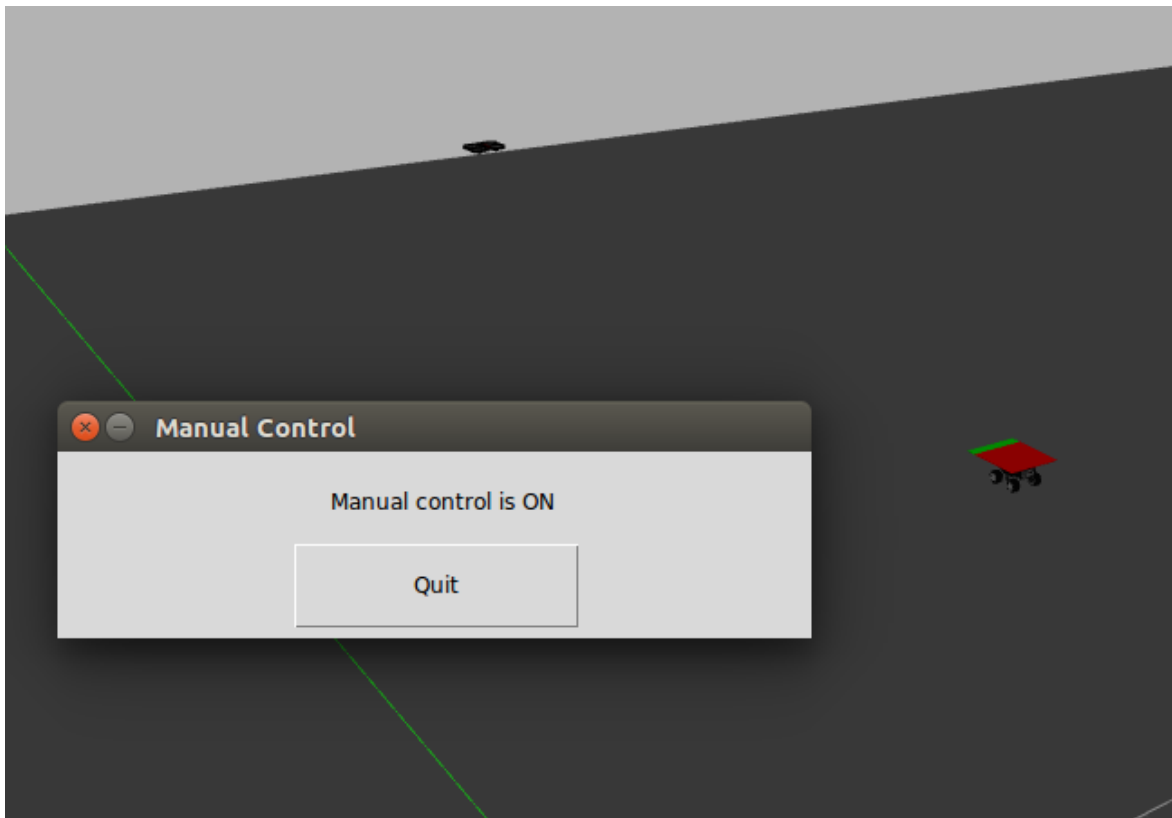


Figure 4.7 The Gazebo simulation and "Manual Control" menu

### **4.3. Conclusion**

Four tests were held: the test of the "Landing Mode", the test of the "Emergency Mode" and the test of the "Manual Mode". The "Emergency Mode" was tested twice: the first time it was launched from the "Flight Management" menu and the second time it was triggered by changing the parameter "Drone Battery Charge" to the value of 10%. All the test were successfully done using the algorithm, described in this work.

The results of the test of the "Landing Mode" also contain the description of the ROS nodes and ROS topics, that were created. Also the description of their communication was provided. Description of every test is provided with the screenshot of Gazebo after finishing the performance of the certain module.

## **5. CONCLUSIONS AND FUTURE RECOMMENDATIONS**

This chapter concludes the obtained results and gives the recommendations for the future improvements.

### **5.1. Conclusion**

The purpose of this work is to design a drone moving target landing system and algorithm for semi- and full autonomous landing control. The landing module was described in Chapter 3 and tested in Chapter 4. The algorithm for the landing control is presented in a form of the number of the menus, that allow the user to monitor the flight and perform the landing when it is needed.

The landing control algorithm presented in this work allows to minimize the human factor and to prevent the performance of complicated manoeuvre in the adverse conditions. The algorithm can be applied to any drone model and can be easily changed, if needed. The adverse conditions, listed in Section 2.2, may depend on many things: the drone model, the region, where algorithm is to be used.

The landing algorithm was chosen to be working on any drone, but the UAV should fit in some requirements:

1. The drone should have a bottom camera: no high-resolution is needed.
2. The drone should have IMU sensor.
3. The drone should have an onboard GPS module.

### **5.2. Future recommendations**

Despite the fact that the simulation results are acceptable, there are still some future recommendations:

1. Change the detection method to make the landing algorithm more precise. However the detection method should not set any new requirement for the drone (except for the ones listed in Section 5.1) or for other hardware. The algorithm should remain easy-to-implement.



2. After changing the detection method, make the new landing module get tested with a high-speed landing platform. This will allow to land a drone on a roof of a fast-moving car.

3. To make a full conclusion about the proposed system, it should be tested in a real environment.

## REFERENCES

- [1] J. Crucchiola, "There May Actually Be a Good Reason to Teach Drones to Land on Cars, January 25, 2016. [Online], Available: <https://www.wired.com/2016/01/there-may-actually-be-a-good-reason-to-teach-drones-to-land-on-cars/> [Accessed Dec. 1, 2019]
- [2] B. Carson, "First Look: Uber Unveils New Design For Uber Eats Delivery Drone", Forbes, 2019. [Online]. Available: <https://www.forbes.com/sites/bizcarson/2019/10/28/first-look-uber-unveils-new-design-for-uber-eats-delivery-drone/#52616e7778f2> [Accessed Dec. 10, 2019].
- [3] "Amazon Prime Air," Amazon, 2016. [Online]. Available: <https://www.amazon.com/AmazonPrime-Air/b?ie=UTF8&node=8037720011> [Accessed Dec. 10, 2019].
- [4] Geospatial World. DLR Germany lands UAV on a moving vehicle. (Jan 25, 2016). [Online Video], Available: <https://www.youtube.com/watch?v=TWUM8DcEDhQ> [Accessed Dec. 1, 2019]
- [5] O. Araar, N. Aouf and I. Vitanov. "Vision Based Autonomous Landing of Multirotor UAV on Moving Platform", Journal of Intelligent & Robotic Systems February 2017, Volume 85, Issue 2, pp 369–384.
- [6] A. Borowczyk, D.-T. Nguyen, A. P.-V. Nguyen, D. Q. Nguyen, D. Saussié, and J. Le Ny, "Autonomous landing of a multirotor micro air vehicle on a high velocity ground vehicle," IFAC PapersOnLine, vol. 50, pp. 10488–10494, 2017.
- [7] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, Z. Milosevic, A. Garcia-Vaquero and P. Campoy. "Towards fully autonomous landing on moving platforms for rotary Unmanned Aerial Vehicles", presented at International Conference on Unmanned Aircraft Systems (ICUAS), 2017.
- [8] Y. Feng, C. Zhang, S. Baek , S. Rawashdeh and A. Mohammadi, "Autonomous Landing of a UAV on a Moving Platform Using Model Predictive Control", Drones 2018, 2, 34; doi:10.3390/drones2040034.
- [9] A. M. Almeshal and M. R. Alenezi, "A Vision-Based Neural Network Controller for the Autonomous Landing of a Quadrotor on Moving Targets", Robotics 2018, 7, 71; doi:10.3390/robotics7040071.

- [10] B.-Y. Xing, F. Pan, X.-X. Feng, W.-X. Li, and Q. Gao, "Autonomous landing of a micro aerial vehicle on a moving platform using a composite landmark," *International Journal of Aerospace Engineering*, vol. 2019, 2019.
- [11] D. Falanga, A. Zanchettin, A. Simovic, J. Delmerico, and D. Scaramuzza, "Vision-based autonomous quadrotor landing on a moving platform," in *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pp. 200–207, Shanghai, China, 2017.
- [12] T. Muskardin, G. Balmer, S. Wlach, K. Kondak, M. Laiacker, and A. Ollero, "Landing of a fixed-wing uav on a mobile ground vehicle" in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1237–1242, Stockholm, Sweden, 2016.
- [13] J. Goncalves, J. Lima, and P. Costa, "Real time tracking of an omnidirectional robot - an extended Kalman filter approach," in *Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics - Volume 4: ICINCO*, pp. 5–10, Funchal - Madeira, Portugal, 2015.
- [14] Y.-M. Lee, "System and method for controlling takeoff and landing of drone" U. S. Patent 10,046,856, 14 August 2018.
- [15] D. Banerjee and A. A. Kandhadai, "Systems and methods for landing a drone on a moving base" U. S. Patent 10,152,059, 11 December 2018.
- [16] J. P. G. Moresve "Method and system for controlling the automatic landing/take-off of a drone on or from a circular landing grid of a platform, in particular a naval platform" U. S. Patent 8,626,364, 7 January 2014.
- [17] A. J. Hawkins, "Wing's delivery drones take flight for the first time in Virginia", *The Verge*, 2019 [Online]. Available: <https://www.theverge.com/2019/10/18/20921310/wings-delivery-drones-virginia-first-flight> [Accessed Dec. 10, 2019].
- [18] N. Libby, "How To Find The Best Courier Service For Your Business" , 2017 [Online]. Available: <https://blog.linnworks.com/best-courier-services> [Accessed Dec. 10, 2019].
- [19] S. Мерков, "Мультикоптеры: что нужно знать, чтобы купить дрон (или вовремя отказаться от покупки)", 2018 [Online]. Available: <https://www.ixbt.com/dv/multicopter-buyers-guide-2018> [Accessed Mar. 3, 2020].
- [20] ROS install [Online]. Available: <https://www.ros.org/install/>. [Accessed May. 1, 2020]

- [21] Ubuntu website [Online]. Available: <https://ubuntu.com/>. [Accessed May. 1, 2020]
- [22] Ubuntu website [Online]. Available: <https://ubuntu.com/tutorials/tutorial-install-ubuntu-desktop#1-overview>. [Accessed May. 1, 2020]
- [23] Wiki ROS [Online]. Available: <http://wiki.ros.org/kinetic/Installation/Ubuntu>. [Accessed May. 1, 2020]
- [24] Wiki ROS [Online]. Available: [http://wiki.ros.org/catkin/Tutorials/create\\_a\\_workspace](http://wiki.ros.org/catkin/Tutorials/create_a_workspace). [Accessed May. 1, 2020]
- [25] GitHub – hector\_quadrotor [Online]. Available: [https://github.com/tu-darmstadt-ros-pkg/hector\\_quadrotor](https://github.com/tu-darmstadt-ros-pkg/hector_quadrotor). [Accessed May. 1, 2020]
- [26] GitHub - uav-autonomous-landing [Online]. Available: <https://github.com/pablorpalafox/uav-autonomous-landing>. [Accessed May. 1, 2020]
- [27] OpenCV Documentation [Online]. Available: [https://docs.opencv.org/2.4/modules/calib3d/doc/camera\\_calibration\\_and\\_3d\\_reconstruction.html](https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html). [Accessed May. 1, 2020]
- [28] Garzón, M.; Garzón-Ramos, D.; Barrientos, A.; Cerro, J.D. Pedestrian Trajectory Prediction in Large Infrastructures. In Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics, Lisbon, Portugal, 29–31 July 2016; pp. 381–389.
- [29] Sony DualShock 4 Gamepad PlayStation 4 Analogue / Digital Bluetooth Black [Online]. Available: <https://nl.icecat.biz/en-sg/p/sony/cuh-zct2/gaming+controllers-dualshock+4-36468958.html>. [Accessed May. 1, 2020]

## APPENDICES

### Appendix 1 The start menu

```

from sys import *
from Tkinter import *
from management import *
import roslaunch
import rospy
class main():
    def __init__(self, master):
        self.master = master
        self.master.geometry("153x49+426+169")
        self.master.resizable(width=False, height=False)
        self.master.title("Start")
        self.master.configure(highlightcolor="black")
        #Quit Button
        QUIT=Button(self.master,text=u'Quit',activebackground="#f9f9f9",com
mand=self.evCancel)
        QUIT.place(relx=0.065, rely=0.204, height=28, width=57)
        #Start Button
        START=Button(self.master,text=u'Start',activebackground="#f9f9f9",co
mmand=self.mCreateManagement)
        START.place(relx=0.523, rely=0.204, height=28, width=66)
        self.master.mainloop()

#Quiting function
def evCancel(self):
    self.master.destroy()

#Starting the simulation function
def mCreateManagement(self):
    uuid = roslaunch.rutil.get_or_generate_uuid(None, False)
    roslaunch.configure_logging(uuid)
    launch
    roslaunch.parent.ROSLaunchParent(uuid,["/interface/launch/start.launch"])
    launch.start()
    self.management=management(self.master)

root = Tk()
main(root)

```

## Appendix 2 The start launch file

```

<launch>
  <arg name="debug" default="false"/>
  <arg name="gui" default="true"/>
  <!-- The world is launched -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find
summit_xl_gazebo)/worlds/summit_xl.world"/>
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="headless" value="false"/>
    <gui>
      <camera name="user_camera">
        <pose>0 0 0 0 0 0</pose>
      </camera>
    </gui>
  </include>
  <!-- Launch the Summit XL model in simulation -->
  <group ns="summit_xl">
    <param name="tf_prefix" value="summit_xl"/>
    <param name="robot_description" command="$(find xacro)/xacro.py '$(find
summit_xl_description)/robots/summit_xl.urdf.xacro'" />
    <!-- Created a node for joints -->
    <node name="summit_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher"
      respawn="false" output="screen">
      <remap from="/joint_states" to="/summit_xl/joint_states" />
    </node>
    <!-- Call a python script to the run a service call to gazebo_ros to spawn a URDF
robot -->
    <arg name="x" default="0"/>
    <arg name="y" default="0"/>
    <arg name="z" default="0"/>
    <arg name="R" default="0"/>
    <arg name="P" default="0"/>

```

```

    <arg name="Y" default="0"/>
    <node name="spawn_summit_xl" pkg="gazebo_ros" type="spawn_model"
respawn="false" output="screen"
      args="-param robot_description
        -urdf
        -x '$(arg x)'
        -y '$(arg y)'
        -z '$(arg z)'
        -R '$(arg R)'
        -P '$(arg P)'
        -Y '$(arg Y)'
        -model summit_xl " />
  </group>
  <!-- Joint controller configurations are loaded from YAML file to parameter server -->
  <rosparam file="$(find summit_xl_control)/config/summit_xl_control.yaml"
command="load"/>
  <node name="controller_spawner" pkg="controller_manager" type="spawner"
respawn="false"
  output="screen" ns="/summit_xl" args="--namespace=/summit_xl
    joint_blw_velocity_controller
    joint_brw_velocity_controller
    joint_frw_velocity_controller
    joint_flw_velocity_controller
    joint_read_state_controller
  "/>
  <!-- Launch the drone model in simulation -->
  <include file="$(find cvg_sim_gazebo)/launch/spawn_quadrotor.launch" >
    <arg name="model" value="$(find
cvg_sim_gazebo)/urdf/quadrotor_sensors.urdf.xacro"/>
  </include>
  <!-- Start the groundtruth state publisher -->
  <node name="gt_altitude_ardrone" pkg="takeoff" type="robot_state_broadcaster"
/>
  <!-- Launch summit's move -->
  <include file="$(find summit_moves)/launch/summit_moves.launch" />
</launch>

```

## Appendix 3 The flight management menu

```
from sys import *

from Tkinter import *

import management_support

import tkMessageBox as msg

import start_support

from manual import *

from landing import *

from emergency import *

import roslaunch

import rospy

import rospkg

import genmsg

import roslib.message

import rosbag

import yaml

from std_msgs.msg import *

#setting the parameters

rospy.set_param('wind_speed', 0.6)

rospy.set_param('drone_speed', 8)

rospy.set_param('control_battery', 50)

rospy.set_param('drone_battery', 80)

rospy.set_param('temp', 15)

class management():

    def __init__(self, master):

        self.win=Toplevel(master)
```



```

self.win.title(u'Flight Management')

self.win.geometry("570x256+423+236")

self.win.resizable(width=False, height=False)

self.win.configure(highlightcolor="black")

#Manual mode Button

MANUAL=Button(self.win,text=u'Manual
Mode',activebackground="#f9f9f9",command=self.mCreateManual)

MANUAL.place(relx=0.035, rely=0.781, height=48, width=169)

#Landing mode Button

LANDING=Button(self.win,text=u'Landing
Mode',activebackground="#f9f9f9",command=self.mCreateLanding)

LANDING.place(relx=0.354, rely=0.781, height=48, width=169)

#Emergency mode Button

EMERGENCY=Button(self.win,text=u'Emergency
Mode',activebackground="#f9f9f9",command=self.mCreateEmergency)

EMERGENCY.place(relx=0.674, rely=0.781, height=48, width=169)

#Quit Button

QUIT=Button(self.win,text=u'Quit',activebackground="#f9f9f9",comman
d=self.evCancel)

QUIT.place(relx=0.86, rely=0.039, height=48, width=69)

#Wind speed

MessageWind=Label(self.win,text=str(rospy.get_param('wind_speed'))
+' m/s',width=125)

MessageWind.place(relx=0.474, rely=0.039, relheight=0.121,
relwidth=0.219)

#Drone speed

MessageDroneSpeed=Label(self.win,text=str(rospy.get_param('drone_s
peed')) +' m/s',width=125)

```

```
MessageDroneSpeed.place(relx=0.474, rely=0.156, relheight=0.121,
relwidth=0.219)
```

### #Sensors

```
MessageSensors=Label(self.win,text=str(rospy.get_param('/use_sim_time')),width=125)
```

```
MessageSensors.place(relx=0.474, rely=0.273, relheight=0.121,
relwidth=0.219)
```

```
MessageSensors.after(1000, self.refresh_label)
```

### #Controller Battery

```
MessageConBattery=Label(self.win,text=str(rospy.get_param('control_battery')) + '%',width=125)
```

```
MessageConBattery.place(relx=0.474, rely=0.391, relheight=0.121,
relwidth=0.219)
```

### #Drone Battery charge

```
MessageBattery=Label(self.win,text=str(rospy.get_param('drone_battery')) + '%',width=125)
```

```
MessageBattery.place(relx=0.474, rely=0.508, relheight=0.121,
relwidth=0.219)
```

### #Temperature

```
MessageTemprature=Label(self.win,text=str(rospy.get_param('temp')) + ' degrees',width=125)
```

```
MessageTemprature.place(relx=0.474, rely=0.625, relheight=0.121,
relwidth=0.219)
```

### #Wind speed Label

```
LabelWind=Label(self.win,background="#d9d9d9",foreground="#000000",relief="flat",anchor='w',justify='left',text=u'Wind speed')
```

```
LabelWind.place(relx=0.125, rely=0.039, height=26, width=103)
```

### #Drone speed Label

```
LabelDroneSpeed=Label(self.win,background="#d9d9d9",foreground="#000000",relief="flat",anchor='w',justify='left',text=u'Drone speed')
```

```
LabelDroneSpeed.place(relx=0.125, rely=0.156, height=26, width=109)
```

```
#Sensors Label
```

```
LabelSensors=Label(self.win,background="#d9d9d9",foreground="#000000",relief="flat",anchor='w',justify='left',text =u'Sensors')
```

```
LabelSensors.place(relx=0.125, rely=0.273, height=26, width=68)
```

```
#Controllers battery charge Label
```

```
LabelOpportunity=Label(self.win,background="#d9d9d9",foreground="#000000",relief="flat",anchor='w',justify='left',text =u'Controller battery charge')
```

```
LabelOpportunity.place(relx=0.125, rely=0.391, height=26, width=168)
```

```
#Battery charge Label
```

```
LabelBattery=Label(self.win,background="#d9d9d9",foreground="#000000",relief="flat",anchor='w',justify='left',text =u'Drone battery charge')
```

```
LabelBattery.place(relx=0.125, rely=0.508, height=26, width=133)
```

```
#Temperature Label
```

```
LabelTemperature=Label(self.win,background="#d9d9d9",foreground="#000000",relief="flat",anchor='w',justify='left',text =u'Temperature')
```

```
LabelTemperature.place(relx=0.125, rely=0.625, height=28, width=97)
```

```
self.win.grab_set()
```

```
self.win.focus_set()
```

```
self.win.wait_window()
```

```
def evCancel(self):
```

```
self.win.destroy()
```

```
def mCreateManual(self):
```

```
uuid = roslaunch.rutil.get_or_generate_uuid(None, False)
```

```
roslaunch.configure_logging(uuid)
```

```
launch =
```

```
roslaunch.parent.ROSLaunchParent(uuid,["/interface/launch/manual.launch"])
```

```
launch.start()
```

```
self.manual=manual(self.win)

def mCreateLanding(self):

    uuid = roslaunch.rlutil.get_or_generate_uuid(None, False)

    roslaunch.configure_logging(uuid)

    launch =
roslaunch.parent.ROSLaunchParent(uuid,["/interface/launch/landing.launch"])

    launch.start()

    self.landing=landing(self.win)

def mCreateEmergency(self):

    uuid = roslaunch.rlutil.get_or_generate_uuid(None, False)

    roslaunch.configure_logging(uuid)

    launch =
roslaunch.parent.ROSLaunchParent(uuid,["/interface/launch/emergency.launch"])

    launch.start()

    self.emergency=emergency(self.win)

def refresh_label(self):

    self.MessageSensors.configure(text=str(rospy.get_param('/use_sim_time')))

    self.MessageSensors.after(1000, self.refresh_label)
```

## Appendix 4 The landing mode file

```

from sys import *
from Tkinter import *
import management_support
import tkMessageBox as msg
import landing_support
class landing():
    def __init__(self, win):
        self.gin=Toplevel(win)
        self.gin.title(u'Landing Mode')
        self.gin.geometry("417x103+467+142")
        self.gin.resizable(width=False, height=False)
        self.gin.configure(highlightcolor="black")
        #Text
        LabelLanding=Label(self.gin,background="#d9d9d9",foreground="#000000",relief="flat",anchor='center',justify='center',cursor="fleur",text=u'Landing has been performed succesfully')
        LabelLanding.place(relx=0.048, rely=0.097, height=36, width=377)
        #Quit Button
        Quit=Button(self.gin,text=u'Quit',activebackground="#f9f9f9",command=self.evCancel)
        Quit.place(relx=0.072, rely=0.485, height=48, width=159)
        #Manual Button
        Manual=Button(self.gin,text=u'ManualMode',activebackground="#f9f9f9",command=self.evCancel)
        Manual.place(relx=0.552, rely=0.485, height=48, width=159)
        self.gin.grab_set()
        self.gin.focus_set()
        self.gin.wait_window()
    def evCancel(self):
        self.gin.destroy()

```

## Appendix 5 The landing mode launch file

```

<launch>
  <node name="platform_detection" pkg="uav_vision" type="platform_detection"
output="screen">
    <param name="use_trackbars" value="false"/>
    <param name="height_landing_platform" value="0.497"
    <param
                                name="platform_position_in_ardrone_topic"
value="/platform/current_platform_position_in_ardrone"/>
    <param
                                name="platform_position_in_world_topic"
value="/platform/current_platform_position_in_world"/>
    <param
                                name="indicator_position_in_ardrone_topic"
value="/platform/current_indicator_position_in_ardrone"/>
    <param name="image_topic" value="/ardrone/bottom/image_raw"/>
    <param name="cam_info_topic" value="/ardrone/bottom/camera_info"/>
    <param name="altimeter_topic" value="/ardrone/altimeter"/>
    <param name="sonar_topic" value="/ardrone/sonar_height"/>
    <param name="gt_altitude_topic" value="/ardrone/groundtruth_altitude"/>
    <param name="imu_topic" value="/ardrone/imu"/>
    <param name="cmd_topic_vel" value="/ardrone/cmd_vel"/>
  </node>
  <param name="/use_sim_time" value="true" />
  <node pkg="ped_traj_pred" type="ped_traj_pred" respawn="false"
name="prediction_kalman_node" output="screen">
    <param
                                name="pose_topic"
value="/platform/current_platform_position_in_world" />
    <param name="path_id" value="1" />
    <!-- For a given path, time between predictions -->
    <param name="time_step" value="0.1" />
    <!-- Maximum predicted time for the given path -->
    <param name="path_time" value="0.1" />
    <!-- Rate at which we predict new paths -->
    <!-- at rate of 2 means we predict a new path every 0.5 seconds -->
    <param name="pub_freq" value="10" />
    <param name="model_covariance" value="2.0" />
    <param name="observation_covariance" value="2.0" />
    <param name="future_path_topic" value="/kalman_prediction_path"/>
  </node> </launch>

```

## Appendix 6 The manual mode file

```

from sys import *

from Tkinter import *

import management_support

import tkMessageBox as msg

import manual_support

class manual():

    def __init__(self, win):

        self.gin=Toplevel(win)

        self.gin.title(u'Manual Control')

        self.gin.geometry("417x103+467+142")

        self.gin.resizable(width=False, height=False)

        self.gin.configure(highlightcolor="black")

        #Text

        LabelManual=Label(self.gin,background="#d9d9d9",foreground="#000000",relief="flat",anchor='center',justify='center',cursor="fleur",text=u'Manual control is ON')

        LabelManual.place(relx=0.192, rely=0.097, height=36, width=267)

        #Quit Button

        Quit=Button(self.gin,text=u'Quit',activebackground="#f9f9f9",command=self.evCancel)

        Quit.place(relx=0.312, rely=0.485, height=48, width=159)

        self.gin.grab_set()

        self.gin.focus_set()

        self.gin.wait_window()

    def evCancel(self):

        self.gin.destroy()

```

## Appendix 7 The manual mode launch file

```
<launch>
  <node pkg="teleop_twist_keyboard" type="teleop_twist_keyboard.py"
name="teleop" output="screen">
  </node>
  <node pkg="ardrone_moves" type="ardrone_moves" name="ardrone_moves"
output="screen">
  </node>
</launch>
```